CONTROL SYSTEM USING PLURAL OBJECTS, AND ITS BUILDING METHOD AND PERIPHERAL DEVICE CONTROL SYSTEM

Patent Number: JP9073395
Publication date: 1997-03-18
Inventor(s): KIMURA YOSHIHIRO; HISAMATSU YUTAKA
Applicant(s):: SEIKO EPSON CORP
Requested Patent: ⌜ JP9073395
Application Number: JP19950229545 19950906
Priority Number(s):
IPC Classification: G06F9/44 ; G06F9/06 ; G06F13/10
EC Classification:
Equivalents:

Abstract

PROBLEM TO BE SOLVED: To provide a control system which can be customized more flexibly and easily as a control system consisting of a plurality of common objects and its building method.
SOLUTION: When a 1st OCX 10 as an instance of a common object generates and controls another 2nd OCX 11, an interface object 15 which makes possible a bilateral communication is generated and utilized. By this interface object 15, an event generated at the 2nd OCX 11 can be sent back to the 1st OCX 10 and the 2nd OCX 11 can be operated completely from the 1st OCX 10.

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平9－73395

(43)公開日　平成９年(1997)３月18日

(51)Int.Cl.⁶		識別記号	庁内整理番号	ＦＩ			技術表示箇所
Ｇ０６Ｆ	9/44	５３０		Ｇ０６Ｆ	9/44	５３０Ａ	
	9/06	５３０			9/06	５３０Ｔ	
	13/10	３２０			13/10	３２０Ａ	Ｃ8-12

審査請求　未請求　請求項の数12　ＯＬ　（全 17 頁）

(21)出願番号　特願平7－229545

(22)出願日　平成７年(1995)９月６日

(71)出願人　000002369
セイコーエプソン株式会社
東京都新宿区西新宿２丁目４番１号

(72)発明者　木村　芳弘
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

(72)発明者　久松　豊
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

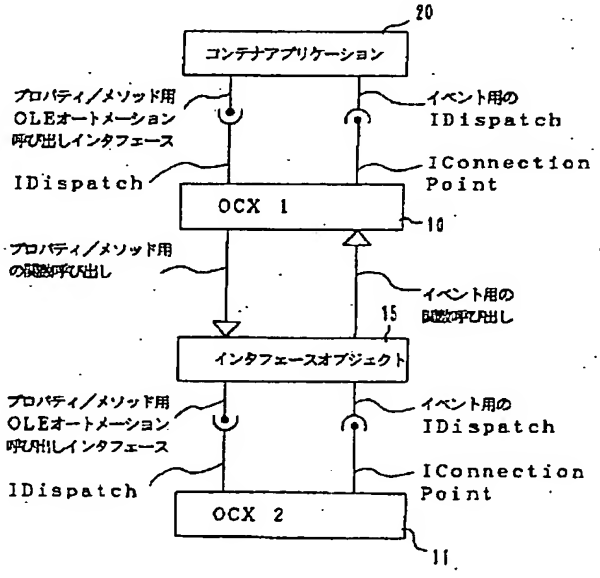
(74)代理人　弁理士　鈴木　喜三郎　（外１名）

(54)【発明の名称】　複数のオブジェクトを用いた制御システム、その構築方法および周辺装置制御システム

(57)【要約】

【課題】　複数のコモンオブジェクトから構成される制御システムにおいて、よりフレキシブルに、また、容易にカスタマイズ可能な制御システムおよびその構築方法を提供する。

【解決手段】　コモンオブジェクトのインスタンスである第１のＯＣＸ１０が、他の第２のＯＣＸ１１を作成し制御する際に、双方向の通信が可能なインタフェースオブジェクト１５を作成し利用可能とする。このインタフェースオブジェクト１５によって、第２のＯＣＸ１１において発生したイベントを第１のＯＣＸ１０に返すことが可能となり、第１のＯＣＸ１０から第２のＯＣＸを完全に動作させることが可能となる。



1

【特許請求の範囲】

【請求項１】　属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかを提供する第１の機能と、非同期に発生したアクションを含むイベントを発信する第２の機能とを備えた複数の制御オブジェクトを有し、

第１の前記制御オブジェクトまたはそのインスタンスによって少なくとも１つの第２の前記制御オブジェクトのインスタンスが作成され、前記第１の制御オブジェクトまたそのインスタンスから前記第２の制御オブジェクトのインスタンスの制御が行われるコンピュータ内のシステムであって、

前記プロパティおよびメソッドの少なくともいずれかを前記第１の制御オブジェクトまたはそのインスタンスと前記第２の制御オブジェクトのインスタンスの間で伝達する機能と、

前記第２の制御オブジェクトのインスタンスから前記第１の制御オブジェクトまたはそのインスタンスに前記イベントを伝える機能とを備えたインタフェースオブジェクトまたはそのインスタンスを有することを特徴とする複数のオブジェクトを用いた制御システム。

【請求項２】　請求項１において、複数の前記第２の制御オブジェクトのインスタンスを有し、それら第２の制御オブジェクトのインスタンス毎に前記インタフェースオブジェクトまたはそのインスタンスが作成され、それぞれの前記インタフェースオブジェクトは前記第１の制御オブジェクトまたはそのインスタンスによって参照可能な独自の識別手段を備えていることを特徴とする複数のオブジェクトを用いた制御システム。

【請求項３】　請求項１において、前記第２の制御オブジェクトは複数の前記イベントを第２の配列に従って発信し、前記インタフェースオブジェクトは前記複数のイベントを第１の配列に変換する手段を備えていることを特徴とする複数のオブジェクトを用いた制御システム。

【請求項４】　請求項３において、前記第２の制御オブジェクトのインスタンス毎に前記インタフェースオブジェクトまたはそのインスタンスが作成され、それぞれの前記インタフェースオブジェクトまたはそのインスタンスは前記第１の配列に従って起動される複数のイベント処理手段を備えていることを特徴とする複数のオブジェクトを用いた制御システム。

【請求項５】　属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかを提供する第１の機能と、非同期に発生するアクションを含むイベントを発信する第２の機能とを備えた複数の制御オブジェクトを有し、第１の前記制御オブジェクトまたはそのインスタンスによって少なくとも１つの第２の前記制御オブジェクトのインスタンスが作成され、これら複数の制御オブジェクトおよびそれらのインスタンスによってコンピュータ内に制御システムを構築する方法であって、

2

前記プロパティおよびメソッドの少なくともいずれかを前記第１の制御オブジェクトまたはそのインスタンスと前記第２の制御オブジェクトのインスタンスとの間で伝達する機能と、前記第２の制御オブジェクトのインスタンスから前記第１の制御オブジェクトまたはそのインスタンスに前記イベントを伝える機能とを備えたインタフェースオブジェクトまたはそのインスタンスを前記第２の制御オブジェクトのインスタンスに対応して作成することを特徴とする制御システムの構築方法。

【請求項６】　請求項５において、それぞれの前記インタフェースオブジェクトに独自の識別手段が付与され、その識別手段が前記イベントと共に前記第１の制御オブジェクトまたはそのインスタンスに伝えられることを特徴とする制御システムの構築方法。

【請求項７】　請求項５において、前記第２の制御オブジェクトは複数の前記イベントを第２の配列に従って発信し、前記インタフェースオブジェクトは第１の配列に従って起動される複数のイベント処理手段を備えており、前記インタフェースオブジェクトまたはそのインスタンスを作成する際に前記第２の配列を第１の配列に変換する第３の配列を作成することを特徴とする制御システムの構築方法。

【請求項８】　アプリケーションシステムに対し周辺装置の管理を行う制御システムであって、

属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかを提供する第１の機能および非同期に発生するアクションを含むイベントを発信する第２の機能を備えた複数の制御オブジェクトまたはそのインスタンスを有し、

これら複数の制御オブジェクトまたはそのインスタンスからなる制御オブジェクト群は、第１の前記制御オブジェクトまたはそのインスタンスと、

これら第１の制御オブジェクトまたはそのインスタンスによって作成された第２の前記制御オブジェクトのインスタンスと、

前記プロパティおよびメソッドの少なくともいずれかを前記第１の制御オブジェクトまたはそのインスタンスと前記第２の制御オブジェクトのインスタンスの間で伝達する機能、および前記第２の制御オブジェクトのインスタンスから前記第１の制御オブジェクトまたはそのインスタンスに前記イベントを伝える機能を備えたインタフェースオブジェクトまたはそのインスタンスとを備えていることを特徴とする周辺装置の制御システム。

【請求項９】　請求項８において、前記第２の制御オブジェクトは、前記周辺装置の第１の周辺装置に対応していることを特徴とする周辺装置の制御システム。

【請求項１０】　請求項９において、前記第２の制御オブジェクトのプロパティは前記第１の周辺装置に固有な仕様に対応した属性値を含み、前記メソッドは前記第１

の周辺装置に固有な命令を含み、さらに、前記イベントは前記第１の周辺装置において非同期に発生するアクションを反映可能であることを特徴とする周辺装置の制御システム。

【請求項１１】　請求項９において、前記第１の制御オブジェクトは、前記インタフェースオブジェクトを介して伝達された前記第２の制御オブジェクトのプロパティ、メソッドおよびイベントの少なくともいずれかを普遍的な仕様に変換することを特徴とする周辺装置の制御システム。

【請求項１２】　請求項９において、前記アプリケーションシステムはＰＯＳ制御システムであり、前記第２の制御オブジェクトはＰＯＳを構成可能な各種の外部周辺装置毎に用意されており、前記ＰＯＳ制御システムは前記第１の制御オブジェクトまたはそのインスタンスを介して前記外部周辺装置を制御することを特徴とする周辺装置の制御システム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、共通に使用できる複数のオブジェクトを備え、これらのオブジェクトを用いてカスタマイズ可能な制御システム、その構築方法および周辺装置制御システムに関するものである。

【０００２】

【従来の技術】近年、コンピュータを用いて多種多様の処理が行われ、コンピュータに接続可能な外部入出力機器も多種多様なものが用意されている。従って、コンピュータを中心として、ユーザーの環境や目的などに合わせてフレキシブルなシステムを構築し、カスタマイズ可能なシステムを提供することが重要となっている。例えば、共通な規格のバスを用いて１つのコンピュータに様々なタイプのプリンター、ディスプレイ、バーコードリーダーを始めその他の入出力機器が接続可能であり、コンピュータにはこれらの入出力機器を制御可能なソフトウェアがインストールされる場合には、多くの機器を的確に制御するように制御システムが構築されていなければならない。また、複数のコンピュータをネットワークで結んだ環境が簡単に構築できるので、個々のコンピュータによって処理すべき対象およびその手順を様々に変更できることが望ましい。

【０００３】このようなフレキシブルなシステムをコンピュータ内に構築する１つの方法として、共通に用いることのできる複数のオブジェクト（コモンオブジェクト）を作成し、これらを活用してオペレーティングシステムやアプリケーションプログラムを構築することが考えられる。マイクロソフト社のＯＬＥ（ＯｂｊｅｃｔＬｉｎｋｉｎｇ　ａｎｄ　Ｅｍｂｅｄｄｉｎｇ）およびＯＬＥプログラミング用に用意されたインタフェースを用いたＯＬＥオートメーションあるいＯＬＥカスタムコントロール（ＯＣＸ）といったシステムはその１つである。

【０００４】

【発明が解決しようとする課題】本発明においては、コモンオブジェクトを活用して多種多様なオペレーティングシステムあるいはアプリケーションプログラムを簡単に構築できるシステムおよびその方法を提供することを目的としている。特に、１つのコモンオブジェクトが他のコモンオブジェクトの機能を充分に活用できるシステムおよびその構築方法を提供することを目的としている。さらに、１つのコモンオブジェクトがクライアントあるいはコントロールとなり、他のコモンオブジェクトをサーバーとして活用する際に、サーバー側で発生したイベントを個別に識別でき、また、その処理手順を記述しやすいシステムおよび構築方法を提供することを目的としている。さらに、多種多様な周辺装置の制御に適したオープンで処理速度の早い制御システムを複数のコモンオブジェクトによって構成可能とすることも本発明の目的の１つである。

【０００５】

【課題を解決するための手段】本発明は、コモンオブジェクトとして用いられるように属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかをアプリケーションプログラム、他の制御オブジェクトあるいはそのインスタンスに提供する第１の機能と、非同期に発生するアクションを含むイベントを発信する第２の機能とを備えた複数の制御オブジェクトを有するシステムを対象としている。このシステムにおいてコンピュータ内に実行可能な制御システムを構築する際に、第１の制御オブジェクトまたはそのインスタンスによって少なくとも１つの第２の制御オブジェクトのインスタンスが作成され、第１の制御オブジェクトまたはそのインスタンスから第２の制御オブジェクトのインスタンスの制御を行うことが考えられる。本発明のこのような制御システムにおいて、プロパティおよびメソッドの少なくともいずれかを第１の制御オブジェクトまたはそのインスタンスと第２の制御オブジェクトのインスタンスの間で伝える機能と、第２の制御オブジェクトのインスタンスから第１の制御オブジェクトまたはそのインスタンスにイベントを伝える機能とを備えたインタフェースオブジェクトを設けるようにしている。従って、本発明により、１つの制御オブジェクトまたはそのインスタンスから、他の制御オブジェクトまたはそのインスタンスをプロパティ、メソッドおよびイベントの全てを含んで双方向の通信が可能なインタフェースオブジェクトおよびそのインスタンスによって連絡することが可能となる。このため、制御オブジェクトとして提供された機能を充分に活用した制御システムを簡単に構築できる。本発明の制御システムにおいては、第１の制御オブジェクトまたはそのインスタンスが、第２の制御オブジェクトまたはそのインスタンスにおいて発生したイベントを受け取ることができ、第１の

制御オブジェクトまたはそのインスタンスがそのイベントに対応した処理を行い、さらに、第１の制御オブジェクトまたはそのインスタンスを活用している他の制御オブジェクトあるいはそのインスタンス、アプリケーションプログラムあるいはオペレーティングシステムに対しイベントを迅速に伝えることが可能となる。従って、本発明により、コモンオブジェクトの機能を完全に活用した制御システムを構築することができる。

【０００６】このようなインタフェースオブジェクトまたはそのインスタンスは、第２の制御オブジェクトのインスタンス毎に作成することが望ましく、複数のインスタンスを作成する際は、それぞれのインスタンスに対応するインタフェースオブジェクトに第１の制御オブジェクトまたはそのインスタンスによって参照可能な独自の識別手段、例えば識別番号を設けることが有効である。各インスタンスにおいて発生したイベントは、対応するインタフェースオブジェクトまたはそのインスタンスを介して第１の制御オブジェクトまたはそのインスタンスに伝えられるので、第１の制御オブジェクトまたはそのインスタンスはいずれの作成したインスタンスにおいて発生したイベントであるかを判別できる。

【０００７】また、第２の制御オブジェクトが複数のイベントを独自の第２の配列に従って発信してもインタフェースオブジェクトおよび第１の制御オブジェクト側を変更しないで済むように、インタフェースオブジェクトに複数のイベントを第１の配列に変換する手段を設けておくことが望ましい。また、第２の制御オブジェクトにおける配列に係わらず所定の第１の配列に変換されるので、イベント毎に起動されるイベント処理手段の記述も容易となる。

【０００８】本発明に係るインタフェースオブジェクトを用いることによって、複数の制御オブジェクトおよびそれらのインスタンスによってコンピューターの機種やそれに接続される周辺装置に対しオープンな制御システムを構築できる。例えば、第２の制御オブジェクトを複数の周辺装置の中の第１の周辺装置に対応して用意することによって、第２の制御オブジェクトに第１の周辺装置に固有な仕様に対応した属性値を含んだプロパティと、命令を含んだメソッドと、さらに、第１の周辺装置において非同期に発生するアクションを反映可能なイベントを設定することができる。そして、第２の制御オブジェクトと第１の制御オブジェクトをインタフェースオブジェクトを介して接続することによって、第１の制御オブジェクトにおいて第２の制御オブジェクトのプロパティ、メソッドおよびイベントの少なくともいずれかを複数の機種、メーカーあるいは複数の種類の周辺装置に対して普遍的に規定された仕様に変換し、周辺装置に依存しない共通のアプリケーションプログラムインタフェース（ＡＰＩ）をアプリケーションプログラムや上位の制御オブジェクトおよびそのインスタンスに対し提供することができる。さらに、第１の制御オブジェクトのＡＰＩをコンピュータの機種に依存しない普遍的なものにすることも可能であり、機種依存性がなく、汎用性の高いアプリケーションプログラムとすることができる。従って、本発明の制御システムにより、ユーザーは自己の環境に合わせたアプリケーションおよびハードウェアによってＰＯＳシステムなどの周辺装置を多く使用するシステムを簡単に、そしてフレキシブルに構築できる。

【０００９】

【発明の実施の形態】

〔ＯＬＥオートメーションをベースにした例〕以下において、マイクロソフト社の提供するプログラム開発環境であるMicrosoft Foundation Class（ＭＦＣ）環境下において本発明を実施した例に基づきさらに詳しく説明する。

【００１０】ＭＦＣはＯＬＥのプログラミングを容易に行えるようにさまざまなライブラリーを提供しており、オブジェクト指向のプログラミング言語であるMicrosoft 社が提供するVisual C++（以下VC++）などによってＯＬＥオートメーション機能を用いたシステムの開発を行うことができる。本例では、図１に示すように、アプリケーション（コンテナアプリ）２０と、システム内に共通に活用できるように用意されたコモンオブジェクトあるいはそのインスタンス（ＯＣＸ）１０および１１とを階層的に結合して構築された制御システムについて説明する。

【００１１】コモンオブジェクトは、他の実行型のソフトウェアを構成するオブジェクトとして提供されても良く、あるいはダイナミックリンクライブラリ（ＤＬＬ）として提供されるオブジェクトであっても良い。コモンオブジェクトを提供するこれらの実行型のソフトウェアやＤＬＬはＥＸＥサーバーあるいはＤＬＬサーバーと呼ぶことができ、これらに対し、コンテナアプリ２０をクライアントあるいはコントロールと呼ぶことができる。ＯＬＥに対応したこれらのコモンオブジェクトは、例えば、ＭＦＣが提供しているＣＣｍｄＴａｒｇｅｔクラスから派生させることができる。ＣＣｍｄＴａｒｇｅｔクラスは、コモンオブジェクトを管理するためのＩＵｎｋｎｏｗｎインタフェースや、その他のコモンオブジェクトとして必要なインタフェース等をサポートするクラスである。ＣＣｍｄＴａｒｇｅｔクラスはコントローラ側からデータを保持した構造体などを受渡しするためのIDispatch インタフェースもサポートしており、このメンバ関数Invokeをコールすることによってコモンオブジェクト側のプロパティあるいはメソッドを用いることができる。

【００１２】プロパティは、そのコモンオブジェクトの有する属性であり、カラー、テキスト、番号、フォントあるいはプッシュボタンがプッシュされた時の動作などが含まれる。メソッドは、コモンオブジェクトにインプ

リメントされた、例えば編集機能などのファンクションであり、メソッドをコールすることによりコモンオブジェクトの機能を操作することができる。コモンオブジェクトは、プロパティやメソッドを提供する機能に加え、コモンオブジェクトに対する外部からのアクション、例えばマウスボタンのクリックやキー入力、その他の非同期に発生するアクションをイベントとして発信する機能を備えることができる。これは通常、コントロール側がプロパティー等をコールするのと同様に、コモンオブジェクトにおいてInvoke関数をコールすることによって行われる。

【００１３】図１には、コンテナアプリ２０があるコモンオブジェクトあるいはそのインスタンスを用いている制御システムを示してある。本図に示したシステムでは、コンテナアプリ２０が第１のオブジェクトのインスタンス（ＯＣＸ）１０を作成し、さらに、その第１のＯＣＸ１０が別の第２のオブジェクトのインスタンス（ＯＣＸ）１１を作成して制御している制御システムを示してある。ＭＦＣライブラリーには、コンテナアプリやＯＣＸ等からInvoke関数をコールする複雑さをカバーするためにCOleDispatchDriverクラスが用意されており、ＯＣＸ１０はこのクラスから派生させたオブジェクトのメンバー関数をコールすることにより、他の制御オブジェクトのインスタンスであるＯＣＸ１１を作成し、ＯＣＸ１１のメソッド等を活用できる。

【００１４】サーバー内のコモンオブジェクトを用いる場合、例えば、コンテナアプリの動作しているプロセッサーと異なるプロセッサーでそのコモンオブジェクトが動作しているケースも考えられる。このようなケースでは、本例のような制御システムを、コントロール側とサーバー側で通信処理を行うことにより構築することができる。また、１つのプロセッサーに活用するコモンオブジェクトをコピーして制御システムを構築することも可能である。いずれの場合も、用いられるコモンオブジェクトのコード自体には変更は全く加えられず、そのコモンオブジェクトを活性化する毎にデータやスタックなどの領域が用意され、それを用いてコモンオブジェクト（サーバー側）とコントロール側が通信しながら制御システムを構築する。従って、同一のコモンオブジェクトをコントロール側で複数用いることも可能であり、この場合、コモンオブジェクトが複数コピーされるのではなく、データやスタックなどの領域がそれぞれに用意される。このようにコモンオブジェクトが他のオブジェクト等によって活用可能な状態になることを本明細書ではインスタンスと称する。

【００１５】さて、第１のＯＣＸ１０がコントロール側となって第２のＯＣＸ１１を呼び出して使用したい場合、VC++のクラスウィザードを用いてCOleDispatchDriverクラスから派生されたインタフェースオブジェクトを用いることが可能であることは上述した通りである。なお、以下において、インタフェースオブジェクトとは、そのインスタンスも含めた意味で用いている。VC++2.0のCOleDispatchDriverクラスは、メソッドやプロパティをIDispatch インタフェースを介して第２のＯＣＸに渡す機能をサポートしているが、第２のＯＣＸが発信するイベントをとらえる機能をサポートしていない。従って、コンテナアプリ２０の側が第１のＯＣＸ１０の用いる第２のＯＣＸ１１を識別しており、第２のＯＣＸのイベントを受信する機能を予め備えている場合は何ら問題はない。しかしながら、コンテナアプリ２０の側が第１のＯＣＸ１０の用いる第２のＯＣＸ１１を識別していない場合は第２のＯＣＸ１１で発生したイベントをとらえることができない。

【００１６】用意されているコモンオブジェクトが予め判明していたり、コンテナアプリに合わせてコモンオブジェクトが用意されている場合は、コモンオブジェクトがコンテナアプリに対してイベントを渡せるようにシステムを構築したり、あるいは、第２のＯＣＸではコンテナアプリに影響を与えるようなイベントを発生させないなどの処理が可能であろう。しかしながら、種々の外部入出力端末がサポートされるなどコンピュータの環境が様々に変化し得る今後の制御システムを考慮すると、よりフレキシブルな制御システムを構築できることが望ましい。このためには、コンテナアプリ２０がコモンオブジェクトを活用してカスタマイズが可能なように、コモンオブジェクトもコモンオブジェクトを用いたカスタマイズが可能な環境をサポートすることが考えられる。従って、本発明では、コモンオブジェクトからコモンオブジェクトに双方向の通信が可能なインタフェースオブジェクト１５を提供し、さらにフレキシブルな制御システムを簡単に構築できるようにしている。双方向の通信が可能なインタフェースオブジェクトを用いて、コモンオブジェクト間を結べるようになれば、コンテナアプリ２０の側は第２のＯＣＸ１１を認識する必要は全くなくなり、また、第２のＯＣＸ１１もコンテナアプリ２０に左右されないオブジェクトとして提供することができる。

【００１７】本例においては、ＯＣＸからＯＣＸを使用するためのディスパッチ処理をイベントも含めて完全にサポートされたインタフェースオブジェクトを実現するために、COcxDispatchDriverクラスを提供している。このCOcxDispatchDriverは、CCmdTargetクラスと、COleDispatchDriverクラスとを多重継承(Multiple Inheritance)して派生したオブジェクトクラスであり、次のようになる。

【００１８】class COcxDispatchDriver:public CCmdTarget,public COleDispatchDriver { .... } ;CCmdTargetクラスは上述したように、サーバーとしての機能を備えたオブジェクトを派生するクラスであり、IDispatch インタフェースのサポートされたクラスである。従って、第２のＯＣＸに対してサーバー側として動作しイベ

ントを受け取るインタフェースをサポートしたインタフェースオブジェクトを実現できる。一方、COleDispatch Driverは、第２のＯＣＸに対してクライアントあるいはコントロール側として動作し、プロパティ／メソッドに対するアクセスを簡単に行なえる機能を備えたオブジェクトクラスである。従って、クラスウィザードにより上記の２つのクラスから派生させたオブジェクトクラスであるCOcxDispatchDriverクラスは、上記の２つの主な機能を備え、これらに加えそれぞれのクラスの機能を全て＊継承し、サポートされたオブジェクトクラスなので、本発明に係るインタフェースオブジェクトを生成するオブジェクトクラスとして適している。CCmdTargetクラスとCOleDispatchDriverクラスには重複する部分がないため、全く問題なく多重継承が可能である。

【００１９】このように派生されたCOcxDispatchDriverクラスの仕様は以下の通りである。

【００２０】

```
Class COcxDispatchDriver : public CCmdTarget,
                           public COleDispatchDriver
COcxDisptchDriver::COcxDispatchDriver()  //  生成
public:  //  メンバ変数
UINT m __ObjID;        //   オブジェクトインスタンスのID
                         EstablishConnection() でユニークに初期化される
IID  m __IIDEvents;  //    イベント  インタフェース ID
UNIT m __nEvents;    //    イベント数
CDWordArray m__dispID  //  ユーザが定義したディスパッチマップに対応して
                         ディスパッチIDを変換するための配列
public:  //  メンバ関数
Bool EstablishConnection(
          REFCLISD clsid,
          COleException* pError=NULL); //    オブジェクトインスタンス
                         を作成し、メソッド、プロパティ、イベントへの
                         アクセスコネクションを有効にする
void DestroyConnection();  //  コネクションを開放し、オブジェクト
                         インスタンスを削除する
```

COcxDispatchDriverクラスでは、汎用的なクラスを実現するためにイベントの処理関数（イベントハンドラ）は含んでいない。従って、COcxDispatchDriverから派生されたオブジェクトに特定のＯＣＸのイベントに対応するイベントハンドラを記述する必要がある。まず、ＯＣＸの返すイベントの外部名に合わせてディスパッチマップを記述する。ディスパッチマップの各イベントの順番は任意で良く、後述するようにイベントとのコネクションをセットするときに実際に受信するイベントの順番との対応付けがなされる。ただし、ＯＣＸの全部のイベントを完全に記述しておく必要がある。

【００２１】ディスパッチマップのエントリは例えば以下のようになり、マクロのパラメータは、順に、ディスパッチドライバクラス名、外部イベント名、イベントハンドラ名、戻り値、パラメータ情報となる。

【００２２】DISP __FUNCTION(__DSoprn, "ControlCompleteEvent", ControlCompleteEvent, VT __EMPTY, VTS __I4 VTS__SCODE VTS __PBSTR VTS __I4)このような外部イベント名に合わせたディスパッチマップを記述することにより、ＯＣＸで発生したイベントを直観的に判りやすい形式で受け取ることができる。さらに、イベントハンドラのプロトタイプ宣言は下記のようになるので、通常の関数の形式で記述でき、プログラム開発が極めて容易となる。

【００２３】void __DSoprn::ControlCompleteEvent(long ControlID, SCODE Result, BSTR FAR*, pString, long data);次に、COcxDispatchDriverクラスから派生され、上記のようにインタフェースの準備されたインタフェースオブジェクト１５と第２のＯＣＸ１１とのコネクションをセットするステップを図２および図３に示したフローチャートに基づき説明する。本例のインタフェースオブジェクト１５においては、そのメンバ関数COcxDispatchDriver::EstablishConnection() を呼び出すだけで、第２のＯＣＸであるオブジェクトインスタンスが作成され、プロパティーおよびメソッドを提供し、イベントを受け取るコネクションがセットされる。

【００２４】図２に示すように、EstablishConnectionがコールされると、ステップ３１において、clsid によって指定された第２のＯＣＸであるオブジェクトインスタンス１１が生成され、動作を開始する。変数clsid は作成するＯＣＸを識別するための１２８ビットの値が格納されている。ステップ３２において第２のＯＣＸ１１が動作を開始すると、ステップ３３においてこのインスタンス１１のIDispatch インタフェースを取得し保存する。インタフェースオブジェクト１５は、このIDispatch を通じてInvoke関数をコールし、インスタンス１１に

プロパティまたはメソッドを渡すことができる。ステップ３４において、第２のＯＣＸ１１にプロパティあるいはイベントを渡すコネクションがセットされると、ステップ３５において、第２のＯＣＸ１１のイベントのインタフェースがチェックされる。

【００２５】ステップ３６において、イベントに関するインタフェースがあるとステップ３７において、イベントに関する情報を格納する。第２のＯＣＸ１１において用意されているイベント名リストはEntryNamesに、ディスパッチＩＤリストはDispIDs に、また、パラメータ情報リストはParamInfo に格納される。ステップ３８においてこれらの処理が通常に終了すると、図３に示したイベントの対応付けを行う工程に移行する。一方、コネクションがセットできない場合はステップ３９においてエラー処理を行う。

【００２６】図３にイベントの対応付けを行う工程を示してある。まず、ステップ４１においてカウンタfcntをゼロクリアーする。次にステップ４２においてカウンターfcntを確認し、ステップ４３において第２のＯＣＸ１１から得られたイベント名EntryNames(fcnt)がインタフェースオブジェクト１５のイベントハンドラに予め用意したマップpDispMapにあるか否かをチェックする。マップpDispMapにある場合は、さらに、ステップ４４において、パラメータ情報ParamInfo (fcnt)がイベントハンドラに予め用意したマップpDispMapと一致するかを確認する。確認できた場合は、ステップ４５において、第２のＯＣＸ１１から得られたイベントのディスパッチＩＤであるDispIDs (fcnt)をイベントハンドラに予め用意したマップpDispMapでのインデックスに変換するための配列m __dispIDを設定する。ステップ４６において、カウンタfcntをインクリメントし、すべてのイベントに対し、これらのステップを繰り返す。予め用意したマップpDispMapとの対応が付くと、ステップ４７において、第２のＯＣＸ１１のイベントに関するコネクションをセットする。このステップにおいては、インタフェースオブジェクト１５に用意されたイベントを受け取るためのIDispatch のアドレスを、オブジェクト間の結合点を実現するために用意されているインタフェースであるIConnectionPointに対し渡すことによってイベントに関するコネクションをセットする。これによって、第２のＯＣＸ１１においてInvoke関数をコールすると、イベントをインタフェースオブジェクト１５に渡すことができる。すなわち、インタフェースオブジェクト１５が第２のＯＣＸ１１からイベントを受け取ることができる。

【００２７】ステップ４８においてコネクションのセットが無事に終了すると、ステップ４９においてイベントのディスパッチＩＤを変換するための配列m __dispIDを保存する。ステップ５０において、上記のステップが全て正常に終了するとインタフェースオブジェクト１５と第２のＯＣＸ１１とのコネクションのセットが終了し、第１のＯＣＸ１０に対し第２のＯＣＸ１１のプロパティあるいはメソッドを提供可能となり、さらに、第２のＯＣＸ１１から第１のＯＣＸ１０にイベントが伝えられるようになる。実際には、インタフェースオブジェクト１５が第２のＯＣＸ１１からイベントを受け取ったことを第１のＯＣＸ１０に通知する必要がある。従って、第１のＯＣＸ１０にインタフェースオブジェクト１５がイベント受け取った場合にコールされるためのｐｕｂｌｉｃなメンバ関数が用意され、インタフェースオブジェクト１５はそのメンバ関数をコールする。

【００２８】このように、本例のインタフェースオブジェクト１５を用いることによりコモンオブジェクトあるいはそのインスタンス同士の間で双方向の通信が可能となる。従って、コモンオブジェクトあるいはそのインスタンスを用いた、いっそうフレキシブルな制御システムを構築することができる。さらに、本例のインタフェースオブジェクトにおいては、イベントに関するコネクションをセットするときに、コモンオブジェクト側で用意されたイベントの配列と、インタフェースオブジェクト側で用意したイベントの配列を確認し、対応付けるようにしている。従って、第１のＯＣＸ１０に対して第２のＯＣＸ１１のインタフェースが完全に既知となっていなくとも、イベントの種類と数が合致していればコネクションをセットすることができる。すなわち、プロパティ、メソッドおよびイベントの名称が合致していれば複数のコモンオブジェクト間に双方向のコネクションをセットすることができる。このため、コモンオブジェクトの開発が容易となり、バージョンアップが行われたり、異なったオブジェクトが採用された場合であっても、フレキシブルなシステムをより安全、確実に構築することができる。

【００２９】さらに、イベントを受け取ったときに起動されるイベントハンドラも通常の関数の形式で記述することができるので、オブジェクトの開発が極めて容易となり短時間で行えるようになる。

【００３０】図４に、第２のＯＣＸ１１から、インタフェースオブジェクト１５に用意されたIDispatch インタフェースのイベントを通知する呼び出し Invoke(EventID,....) がコールされた場合の処理を示してある。IDispatch インタフェースは Invoke() および、それをサポートする基本的な関数 (AddRef(), Release(),QueryInterface(IID&, LPUNKNOWN*) 等) を提供しており、IDispatch インタフェースが、相手のＯＣＸが発生するイベントの IID (Interface ID) にQueryInterface(...) を介して応答する。 Invoke() がコールされると、この時点でパラメータとしてイベントの種類を規定するdispatch ID やイベントのパラメータを保持する構造体などがインタフェースオブジェクト１５に渡される。インタフェースオブジェクト１５では、ステップ６０においてパラメータが正当か否かをチェックする。正当の場合はス

テップ６１において、第２のＯＣＸ１１から得られたイベントの配列dispIDMemberを用意されたイベントハンドラのマップpDsipMapに合わせて、変換用の配列m __dispIDに基づき変換する。ステップ６２において、dispIDMemberに対応するイベントハンドラをpDsipMapから取り出す。そして、ステップ６３において、イベントハンドラが有効か否かを確認し、有効な場合はステップ６４においてイベントに対応するイベントハンドラを呼び出し、イベントハンドラに定義された処理、例えば、第１のＯＣＸ１０にイベントを発信するなどの処理を行う。

【００３１】図５に、第１のＯＣＸ１０から第２のＯＣＸの複数のインスタンス１１ａおよび１１ｂが作成され、第１のＯＣＸ１０がこれらの第２のＯＣＸ１１ａおよび１１ｂを制御するシステムを示してある。すなわち、第１のＯＣＸ１０は、２つの第２のＯＣＸ１１ａおよび１１ｂに対しクライアントあるいはコントローラ側となっている。このような制御システムにおいて、第１のＯＣＸ１０とこれらのＯＣＸ１１ａおよび１１ｂのインタフェースは、それぞれの第２のＯＣＸ１１ａおよび１１ｂを作成したインターフェスオブジェクト１５ａおよび１５ｂをそれぞれ介して行われる。本例においては、COcxDispatchDriverから派生したインタフェースオブジェクトが生成される度に、個々のインタフェースオブジェクトを識別可能なＩＤが用意される。そして、インタフェースオブジェクトからイベントが第１のＯＣＸ１０に伝えられるときは、個々のインタフェースオブジェクトに固有のＩＤが第１のＯＣＸ１０の側から参照できるようになっている。このＩＤは、インタフェースオブジェクト作成時にユニークな値が割り当てられるようになっているが、必要であればインタフェースオブジェクトを作成した側、すなわち、本例であれば第１のＯＣＸで適当な値に変更することもできる。

【００３２】図５に示したような複数の第２のＯＣＸ１１ａおよび１１ｂが作成された場合、個々の第２のＯＣＸにおいて発生したイベントはインタフェースオブジェクト１５ａおよび１５ｂをそれぞれ介して第１のＯＣＸに伝えられる。従って、第１のＯＣＸは、各々のインタフェースオブジェクトに付された固有のＩＤによっていずれの第２のＯＣＸで発生したイベントであるか明確に識別することができる。このように、本例のインタフェースオブジェクトにより、あるコモンオブジェクトあるいはそのインスタンス（第１のＯＣＸ）から複数のインスタンス（第２のＯＣＸ）を作成して活用することが可能である。そして、このようなケースであっても作成した側（第１のＯＣＸ）は個々のインスタンス（第２のＯＣＸ）を識別し管理しなくとも、どのインスタンス（第２のＯＣＸ）でイベントが発生したかを識別することができる。さらに、イベントハンドラが、インスタンス（第２のＯＣＸ）を発生させたオブジェクトあるいはインスタンス（第１のＯＣＸ）に含まれるのではなく、個々のインタフェースオブジェクト毎に用意される。従って、作成した側（第１のＯＣＸ）は、どのインスタンス（第２のＯＣＸ）で、どのようなイベントが、どのようなパラメータを伴って発生したかを確実に把握することができる。

【００３３】一方、作成されたインスタンス（第２のＯＣＸ）の側も、発生したイベントをそれぞれのインタフェースオブジェクトに単に発信すれば良く、インスタンス自身が最初のインスタンスであるか、あるいは他に同じクラスのインスタンスが存在するのか否か等の判断をする必要はない。

【００３４】このように、本発明にかかるインタフェースオブジェクトを用いることによって、複数のコモンオブジェクトおよびそれらのインスタンスを備えた制御システムをフレキシブルに、そして簡単に構築することができる。それぞれのコモンオブジェクトには、活用する相手のコモンオブジェクトに関し最小限の情報を用意しておけば良く、また、活用される相手のコモンオブジェクトに関する情報も最小限で済む。従って、個々のコモンオブジェクトのカプセル化をいっそう進めることができる。その一方で、本発明のインタフェースオブジェクトはイベントも含めた双方向の通信機能を備えており、活用する相手のコモンオブジェクトおよびそれらのインスタンスを完全に動作させることができ、それらの備えた機能を最大限に発揮させることができる。

【００３５】なお、本例においては、第１および第２のＯＣＸからなる２層の制御システムを用いて説明しているが、３層あるいはそれ以上の階層構造をもった制御システムであっても同様に構築することができる。また、上記のインタフェースオブジェクトを用いてアプリケーションと第１のＯＣＸとの間のインタフェースを実現することも可能である。アプリケーションプログラムに限らず、オペレーションシステムなどであっても同様に構築できることはもちろんである。また、上記においては、マイクロソフト社の提供するＭＦＣ環境下において本発明のオブジェクトインタフェースを実現した例を用いて説明しているが、ＭＦＣライブラリを用いなくとも、同等の機能を備えたオブジェクトインタフェースを作成することは可能である。さらに、共通して活用できるオブジェクトが用意されたシステム環境であれば、マイクロソフト社のＯＬＥオートメーションでなくとも上記と同様のコモンオブジェクトを用いたシステムを構築できる。このようなコモンオブジェクトを用いたシステムは、１つのプロセッサーを備えたコンピュータ単体で実現することもでき、また、ネットワーク環境で接続された複数のプロセッサーを有するシステムにおいても構築できることは上述した通りである。

【００３６】〔周辺装置の制御システム〕図６に、本発明に係るインタフェースオブジェクトと、複数のコモンオブジェクトおよびそのインスタンスによって周辺装置

15

の制御システムを構築した例を示してある。図６にはパーソナルコンピュータ（パソコン）７０を中心に構成されたＰＯＳシステムを示してある。パソコン７０にはＰＯＳアプリケーションプログラム７１がインストールされており、パソコンのオペレーティングシステム（ＯＳ）１０５の上で動作する。ＯＳ１０５は、キーボードドライバ１０６やモニタディスプレイドライバ１０７等を介してパソコンとして通常必要な周辺装置を制御する機能を備えており、アプリケーションプログラム７１とキーボードあるいはモニタディスプレイ（不図示）との間のデータ転送はＯＳ１０５を介して行われる。

【００３７】ＰＯＳシステムには、パソコンに通常用意されるこれらの周辺機器に加えて、客先に金額等を表示するカスタマディスプレイ１１０、レシート等の印刷を行うレシートプリンタ１１２、チェックなどの印刷を行うスリッププリンタ１１３、さらに、金銭を保管するキャッシュドロワ１１５が必要となり、これらの周辺装置はＲＳ－２３２Ｃポート等の拡張用のポートに接続される。例えば、カスタマディスプレイ１１０がＲＳ－２３２Ｃポートに接続され、カスタマディスプレイ１１０をパススルーしてレシートプリンタ１１２およびスリッププリンタ１１３を備えたプリンタ１１１が接続される。キャッシュドロワ１１５は、プリンタ１１１の下部に設置され、プリンタ１１１の制御機構を介して操作される。これらの周辺装置は多くのメーカーから様々な機種が市販されており、ユーザーは自己の環境に適したものを選択してＰＯＳシステムを構築可能である。しかしながら、メーカーや機種の異なる周辺装置は個々に仕様が異なるので、市販されている全ての周辺装置に適合したアプリケーションプログラムの作成は不可能である。さらに、周辺装置がバージョンアップされると、これに伴い仕様も変更になる。従って、従来は、ユーザーは自己に都合の良い周辺機器によってＰＯＳシステムを構築することが困難なこともあり、さらに、周辺機器がバージョンアップされても新しい機種を構築済のＰＯＳシステムにすぐに適用できるとは限らなかった。

【００３８】これに対し、上述したＯＣＸによって制御システムを構築すると、非常にオープンなシステムを形成できる。従って、どのような機種の周辺機器を用いたＰＯＳシステムであっても簡単に構築できる。また、周辺機器のバージョンアップに対しても簡単に対処できる。例えば、図６に示す本例の周辺機器の制御システム７２は、３つのレベルのＯＣＸを備えている。第１のレベルのＯＣＸとして、レシートプリンタフォーマット変換用ＯＣＸ７３と、スリッププリンタフォーマット変換用ＯＣＸ７４が用意されている。これらのＯＣＸ７３およ７４では、例えば、アプリケーションプログラム７１から送られた売上品リストや合計金額等のデータを所定のフォーマットに配置する処理が行われる。所定のフォーマットは、ＯＣＸ７３あるいは７４の内部に設定されていても良いし、下位レベルの制御用ＯＣＸであるレシートプリンタ制御用ＯＣＸ７５あるいはスリッププリンタ制御用ＯＣＸ７６がプロパティとして有しており、これらの制御用ＯＣＸのプロパティを得て変換用ＯＣＸ７３あるいは７４がフォーマットを設定しても良い。いずれにしろ、アプリケーションプログラム７１は、出力表示されるフォーマットに関係なく、出力用のデータを変換用ＯＣＸ７３あるいは７４に受け渡せば良い。従って、インタフェースの形式を限定でき、汎用性の高いアプリケーションプログラムとして提供することができる。また、アプリケーションプログラムから固有のフォーマットで出力されたデータを、第１レベルのＯＣＸによって下位のＯＣＸに共通するフォーマットに変換することも可能である。このように、ＯＣＸを用いて個別に開発されたアプリケーションプログラムの汎用性を高めることも可能である。

16

【００３９】第２のレベルのＯＣＸとしてレシートプリンタ制御用ＯＣＸ７５、スリッププリンタ制御用ＯＣＸ７６、キャッシュドロワ制御用ＯＣＸ７７およびカスタマディスプレイ制御用ＯＣＸ７８が用いられている。これらのＯＣＸ７５～７８は、アプリケーションプログラムあるいは上位のＯＣＸに対し、所定の仕様のインタフェース（ＡＰＩ）を提供するＯＣＸである。従って、ＰＯＳシステムを構成するプリンタ等の周辺装置のメーカーや機種に係わりなく、アプリケーションプログラムや上位のＯＣＸは所定の仕様でデータを提供すれば良い。このレベルのＯＣＸ７５～７８は、周辺装置固有の仕様が反映された下位のドライバレベルのＯＣＸのプロパティを得て、共通の仕様で入力されたデータを下位のＯＣＸ、すなわち、実際にシステムを構築している周辺装置の仕様に合わせたデータに変換する。

【００４０】第３のレベルのＯＣＸは、プリンタドライバＯＣＸ９１および９２と、キャッシュドロワドライバＯＣＸ９３、およびディスプレイドライバＯＣＸ９４である。これらのＯＣＸ９１～９４は、個々の周辺装置に対応したコモンオブジェクトのインスタンスであり、通常はメーカー毎あるいは機種毎に異なり周辺装置と共に提供されるものである。これらのドライバレベルのＯＣＸ９１～９４は、例えば、最大印字行数や印字行ピッチと言ったプリンタ固有の仕様や設定状態（プリンタステータスと称する）をプロパティとして備えており、これらのプロパティは上位のＯＣＸやアプリケーションプログラムで参照できるようになっている。また、指定された位置に文字列を印字する、すなわち、印字命令を出力するというメソッドや、プリンタステータス等のプロパティなどを備えている。従って、ドライバーレベルである第３レベルのＣＯＸからは、例えば、プリンタ用のＯＣＸであれば、送られた印字位置および印字文字列のデータに基づき、改行量（すなわち行ピッチ）、改行コマンド、印字データおよび印字コマンド、オートカットコ

マンドが所定の順序でポートドライバ１００を介してプリンタに送信される。

【００４１】第３レベルのＯＣＸは、プリンタから送信される処理結果およびエラーステータス等の非同期に発生するアクションも受け取る。そして、これらのアクションの内容を対応する周辺装置の仕様に基づき解釈し、普遍的な形式に変換してイベントとして上位のＯＣＸに返す。本例の制御システムでは、双方向の通信が可能なインタフェースオブジェクト８１ａ、８１ｂ、８２ａ～８２ｄを用いて上位のＯＣＸと下位のＯＣＸを接続しているので、プロパティやメソッドに加え、イベントも伝達される。従って、第２レベルの制御用ＯＣＸ７５～７８はイベントを受けて、そのまま、あるいはさらに普遍的な形式に変換し、インタフェースオブジェクト８２ａ～８２ｄを介して上位のＯＣＸやアプリケーションプログラムに伝達する。これに基づき、アプリケーションシステムや上位のＯＣＸは、ユーザーに対してメッセージを出力したりエラー処理ルーチンを起動するなどの処理を行えるので、本例の制御システムにおいては、非同期に発生するアクションに対して迅速にそして的確な処理を行うことができる。

【００４２】このように、本例の制御システムは、ＯＣＸを用いているのでユーザーの採用する周辺装置に適合したカスタマイズが容易に行えるシステムであり、また、ＯＣＸ間でプロパティ、メソッドおよびイベントの双方向の通信が確保されているので、処理速度が早く的確に処理が行える制御システムである。

【００４３】レシート印字機能を例にとってさらに詳しく説明する。第3レベルのＯＣＸであるプリンタドライバＯＣＸ９１は、レシートプリンタ１１２のプリンタステータスをプロパティとして備えている。また、プリンタ１１２に印字命令を出力するメソッドを備えている。さらに、プリンタ１１２からリアルタイムで送られてくる用紙なし（ランアウト）やカバーオープンといったエラーステータスに対応して非同期にイベントを発生する。

【００４４】アプリケーションプログラム７１からデータが第１レベルのレシートプリンタフォーマット変換用ＯＣＸ７３に渡され、所定のフォーマットに変換されたのち、第２レベルの制御用ＯＣＸ７４に渡される。そして、制御用ＯＣＸ７４がドライバＯＣＸ９１の印刷を実行するメソッドをコールすると、ドライバＯＣＸ９１がレシートプリンタ１１２のプリンタステータスに合致した適当なコマンドおよびデータをレシートプリンタ１１２に対し送信し、レシート印字を実行する。レシートプリンタ１１２は、送信されたコマンドを実行し、所定のステータスをドライバＯＣＸ９１に返す。ドライバＯＣＸ９１は、返されたステータスを解釈し、エラーステータスがアクティブでなければイベントを発生せず、印刷処理は終了する。また、上位のＯＣＸであるレシートプリンタ制御用ＯＣＸ７５は、プリンタステータスのプロパティを参照することによってプリンタ処理の結果を知ることももちろん可能である。これらの通信はインタフェースオブジェクト８２ａを介して行われる。

【００４５】一方、印刷実行のコマンドを実行した結果、あるいは待機中にレシートプリンタ１１２にエラーが発生してエラーステータスがアクティブになると、プリンタ１１２はエラーステータスをドライバＯＣＸ９１に送信する。この場合、ドライバＯＣＸ９１はイベントを発生し、エラーの発生を上位の制御用ＯＣＸ７４にインタフェースオブジェクト８２ａを介して伝達する。制御用ＯＣＸ７４は、イベントに対応して特定の処理を行うことも可能であるし、さらにインタフェースオブジェクト８１ａを介してフォーマット変換用ＯＣＸ７４およびアプリケーションプログラム７１に通知し、注意を喚起することによって所定の処理を行わせることも可能である。

【００４６】これらのプロパティ、メソッドおよびイベントに加え、スリッププリンタ１１３に対応したドライバＯＣＸ９２としては、スリップ印字に特有のプロパティおよびイベントを設けておくことが望ましい。例えば、プロパティにスリップ用紙の有無を加えることによって、上位のＯＣＸやアプリケーションが印刷実行を指示するタイミングの判断が可能となる。また、イベントに用紙挿入検出、用紙終端検出を加えることにより、アプリケーションプログラムなどにスリップ印刷後の処理やエラー処理をスムーズに行わせることができる。

【００４７】キャッシュドロワ制御用ＯＣＸ７７はキャッシュドロワドライバＯＣＸ９３と通信する。キャッシュドロワドライバＯＣＸ９３はキャッシュドロワオープンをメソッドとして備えており、キャッシュドロワのオープン・クローズを含めたキャッシュドロワステータスをプロパティとして備えている。また、イベントとしてキャッシュドロワエラーやキャッシュドロワオープン検出を発生することができる。

【００４８】カスタマディスプレイ制御用ＯＣＸ７８は、ディスプレイドライバＯＣＸ９４と通信し、このドライバＯＣＸ９４は、表示位置や表示文字列を指定した表示命令をメソッドとして備えている。また、プロパティとしては表示桁数や表示色などのディスプレイ仕様を備えており、ディスプレイ制御用ＯＣＸ７８はこのプロパティに沿って表示データを与える。また、カスタマディスプレイはイベントに該当するエラー等の発生は少ないので、イベントを発生する機能を削除することも可能である。

【００４９】このように、本発明に係るインタフェースオブジェクトを採用することにより、複数のＯＣＸを用いて階層的な構造を備え、エラー等を含めた非同期に発生するアクションに対しても迅速に対処可能な制御システムを構築できる。また、本例の制御システムは、パソ

コンに接続される周辺装置に対しフレキシブルに対応できるオープンな制御システムである。例えば、プリンタ１１１は、レシート印字機能１１２と、スリップ印字機能１１３と、さらにドロワ１１５の制御機能とを備えており、制御システムはそれぞれの機能毎にＯＣＸ９１～９３が用意されている。このプリンタ１１１に対し、スリップ印字機能の仕様（例えば、最大印字桁数、実行可能なコマンドセット等）の異なるプリンタに置き換える場合は、スリップ印字機能を制御するドライバ用のコモンオブジェクトを新しいプリンタの仕様に対応するものに置き換えれば良い。周辺装置の制御システムにおいては、置き換えられた新しいドライバ用のコモンオブジェクトからＯＣＸを作成し新しいプリンタに適合した制御システムを自動的に形成する。プリンタがバージョンアップされて仕様が変わった場合でも、同様にドライバ用のコモンオブジェクトを置き換えるだけで良く、その他のコモンオブジェクトやアプリケーションプログラムに変更を加える必要はない。

【００５０】カスタマディスプレイについても同様であり、仕様が変更された場合等はドライバ用のコモンオブジェクトを変更すれば良い。また、バーコードリーダ等の他の周辺装置が加えてＰＯＳシステムが構成される場合は、それに対応した制御用のコモンオブジェクトとドライバ用のコモンオブジェクトがパソコンのシステム内に用意されておれば、アプリケーションあるいは上記のＯＣＸによって、下位のＯＣＸが作成され、バーコードリーダ等を含めた周辺装置の制御システムが形成される。

【００５１】同様に、アプリケーションプログラムとのインタフェースを担当する上位のＯＣＸがパソコンのハード等に起因する仕様の違いを吸収し、アプリケーションに対し共通のインタフェースを与えるものであれば、パソコンのハード等の相違とは無関係に共通のアプリケーションプログラムを用いてシステムを構築することも可能となる。パソコン内に用意されたコモンオブジェクトから制御システムがアプリケーションプログラムによって作成され、制御システムを構成するＯＣＸによってパソコンおよびそれに接続された周辺装置に適合したシステムが自動的に形成される。このように、本発明のＯＣＸを用いた制御システムはオープンなシステムであり、インタフェースの仕様を予め合意しておけば、アプリケーションを供給する側は、パソコンやそれに接続される周辺機器の仕様および接続方法等は一切考慮せずにアプリケーションを開発し供給することができる。従って、短期間で開発が可能となり、安価に提供できる。また、ユーザーもパソコンや周辺装置に係わりなく、自己の目的や環境に適したアプリケーションを自由に選択することができる。

【００５２】一方、周辺装置を提供する側も、供給する周辺装置の仕様に対応したコモンオブジェクトを用意し、例えば、周辺装置と共に供給することによって、パソコンやアプリケーションに限定されない汎用的な周辺装置を提供することができる。従って、ユーザーも自己の目的や環境に適した周辺装置を自由に購入し、システムを構築することができる。

【００５３】さらに、本発明に係るインタフェースオブジェクトを採用することによって、コモンオブジェクトおよびそのインスタンス同士の間で双方向にプロパティ、メソッドおよびイベントの伝達が可能となる。従って、本発明の基づく制御システムでは、システムの応答が遅れたり、システムの機能が限定される等のシステムをオープン化する上での弊害は全くない。また、制御システムの機能アップやバージョンアップも個々の制御オブジェクト毎に行うことができるので、システム開発を効率良く、短期間に行える。また、ユーザーもシステムのグレードアップや変更等を簡単に行えるようになる。

【００５４】なお、上記では、用いられる周辺装置の種類や数が多く、また、ユーザーの環境に合わせて様々な機種の周辺装置が用いられる例としてＰＯＳシステムを説明しているが、ＰＯＳシステムに限定されないことはもちろんである。近年のパソコンを中心としたシステムは、ユーザーの目的や能力に合わせて様々な仕様の周辺装置が組み合わされるようになりつつある。本発明に係る共通する制御オブジェクトを用いた制御システムは、様々なユーザーの要望に対してフレキシブルに対応可能なシステムであり、ＰＯＳシステムに限らず、今後様々なシステムに対して適用可能である。

【００５５】

【発明の効果】以上に説明したように、本発明においては、アプリケーションプログラム間で共通に活用できる制御オブジェクトであるコモンオブジェクトおよびそのインスタンスを用いて制御システムを構築している。さらに、インタフェースオブジェクトによって、これらコモンオブジェクトおよびそのインスタンス同士の間で双方向の通信を可能としている。従って、コモンオブジェクトを活用して多種多様なオペレーティングシステムあるいはアプリケーションプログラムを簡単に構築し、提供することができる。このように本発明により、入出力端末や、処理する対象などにより多種多様に変化するコンピュータを中心とした近年のシステムにおいて、いっそうフレキシブルに、そしてカスタマイズ可能なシステムを実現するのに好適な制御システムおよびその構築方法を提供できる。

【００５６】特に、上述したようなコンピューターの機種やそれに接続される周辺装置の種類および数量がユーザーの環境に応じて多種多様に変化する制御システムを構築するのに本発明は好適である。本発明に係る制御システムを用いることにより、アプリケーションプログラムに対し周辺装置に依存しない共通のアプリケーションプログラムインタフェースを提供可能である。さらに、

21

本発明に係る制御システムは、汎用性の非常に高い制御システムでありながらエラー等の非同期に発生するアクションに対しても迅速にアプリケーションを作動させることが可能である。従って、ユーザーに対し自己の目的に対応した快適な動作環境を容易に構築できる制御システムを提供できる。

【図面の簡単な説明】

【図１】本発明の実施例の制御システムの概略構成を示すブロック図である。

【図２】図１に示す制御システムにおいて、インタフェースオブジェクトとオブジェクトインスタンスとの間でコネクションを確認する過程を示すフローチャートである。

【図３】図１に示す制御システムにおいて、インタフェースオブジェクトとオブジェクトインスタンスとの間でイベントの対応付けを行い、コネクションを張る過程を示すフローチャートである。

【図４】図１に示す制御システムにおいて、オブジェクトインスタンス側からイベントをインタフェースオブジェクトに伝える過程を示すフローチャートである。

22

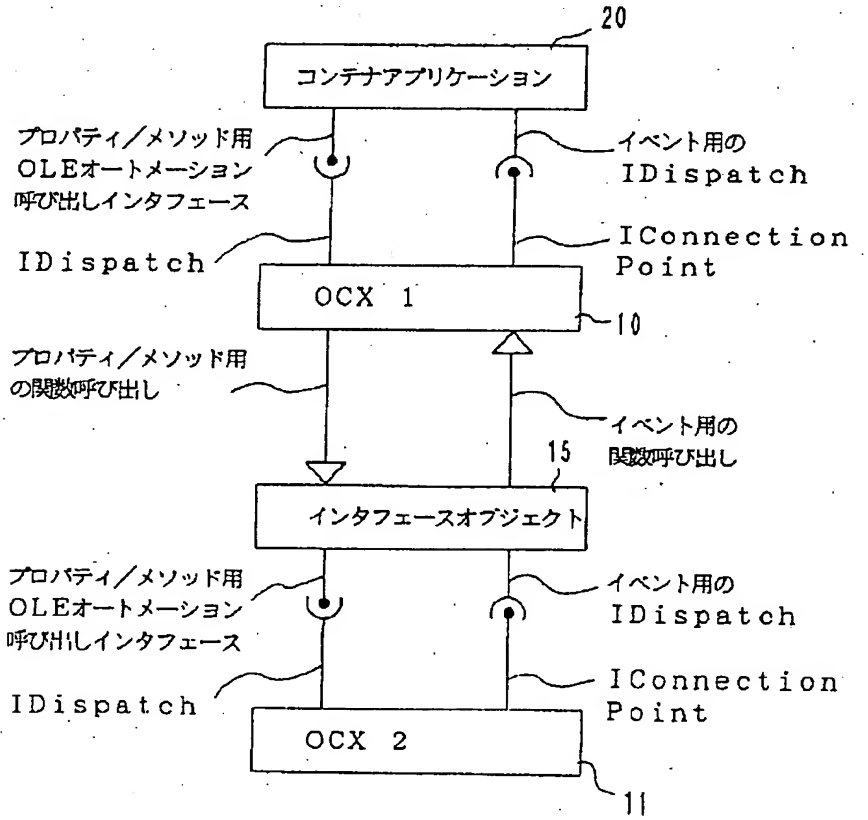
【図５】上記と異なる本発明の実施例の制御システムの概略構成を示すブロック図である。

【図６】本発明の制御システムによって構築されたＰＯＳシステムの周辺装置を制御するシステムを示すブロック図である。

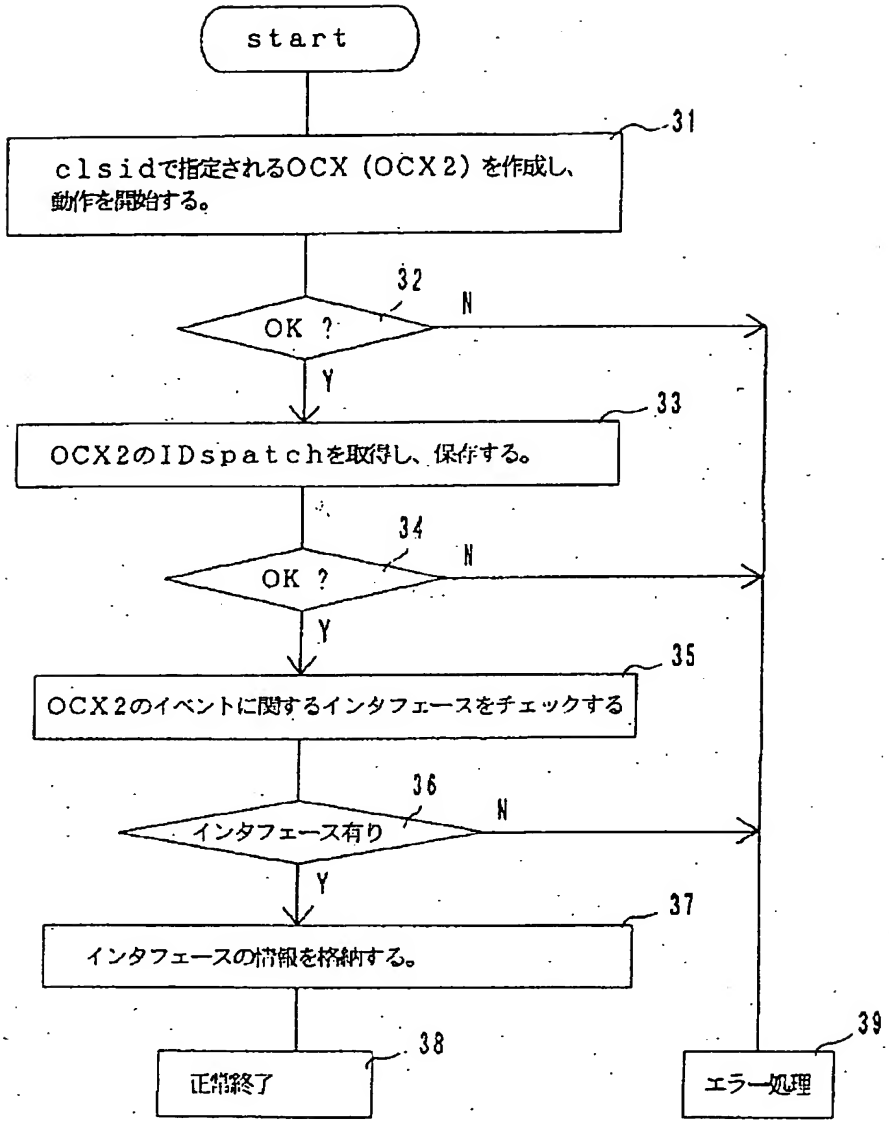
【符号の説明】

１０・・第１のＯＣＸ
１１・・第２のＯＣＸ
１５・・インタフェースオブジェクト
２０・・アプリケーション
７０・・パソコン
７１・・ＰＯＳアプリケーションプログラム
７２・・制御オブジェクトによって構成された制御システム
１００・・ポートドライバ
１１０・・カスタマディスプレイ
１１１・・プリンタ
１１２・・レシート印字機構
１１３・・スリップ印字機構
１１５・・キャッシュドロワ

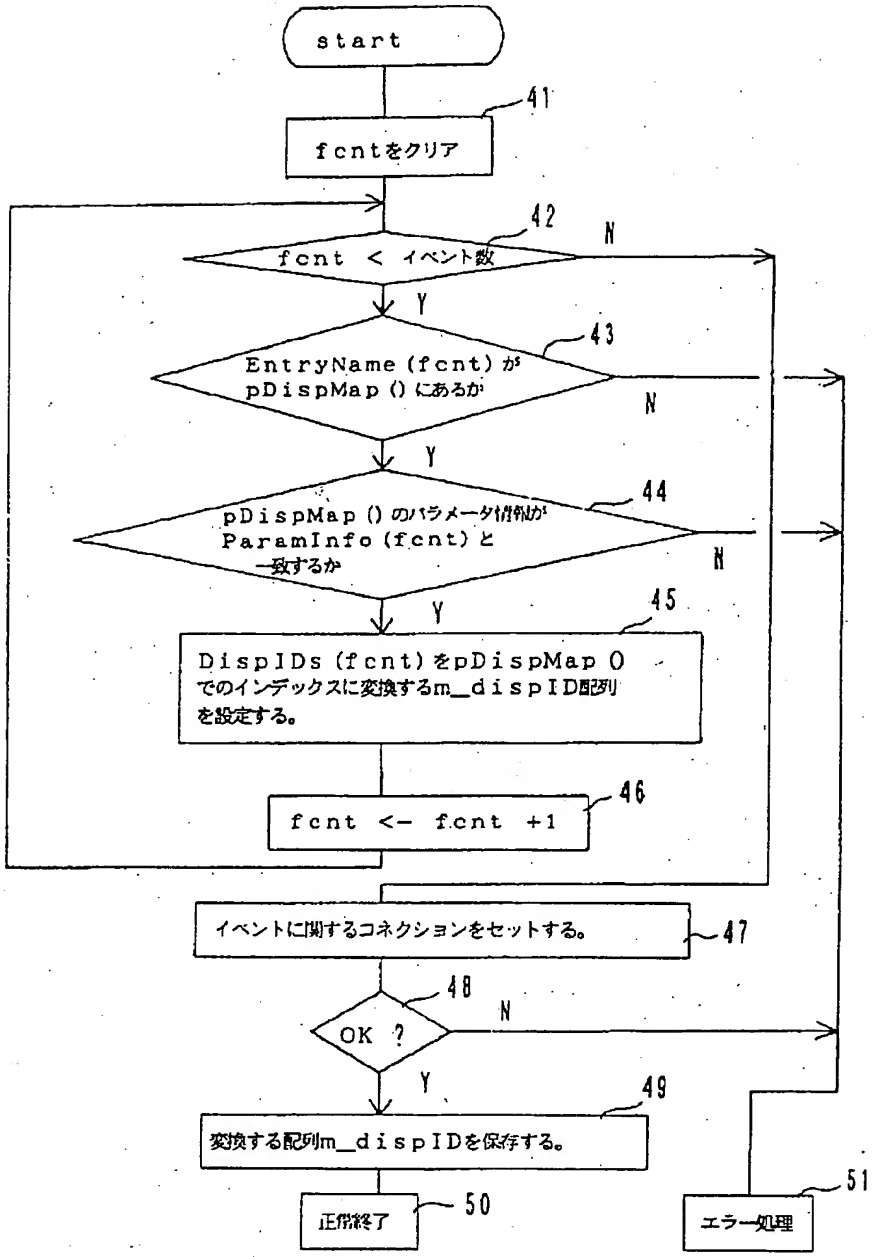
【図１】



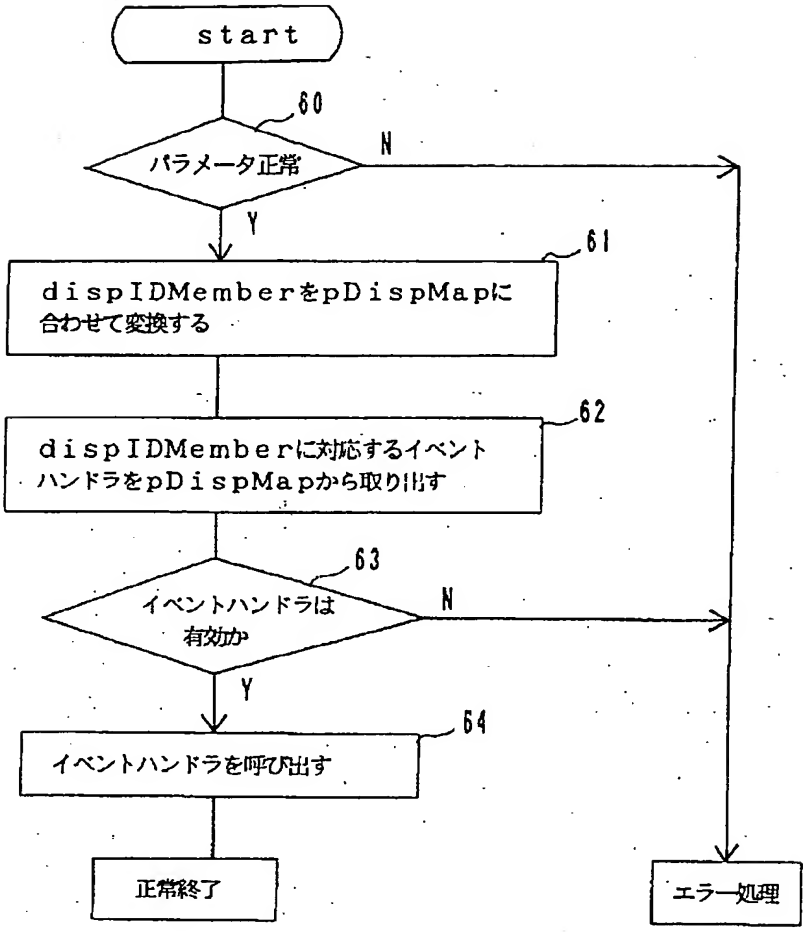
【図２】


start
31
clsidで指定されるOCX（OCX2）を作成し、動作を開始する。
32
OK ?
N
Y
33
OCX2のIDspatchを取得し、保存する。
34
OK ?
N
Y
35
OCX2のイベントに関するインタフェースをチェックする
36
インタフェース有り
N
Y
37
インタフェースの情報を格納する。
38
正常終了
39
エラー処理

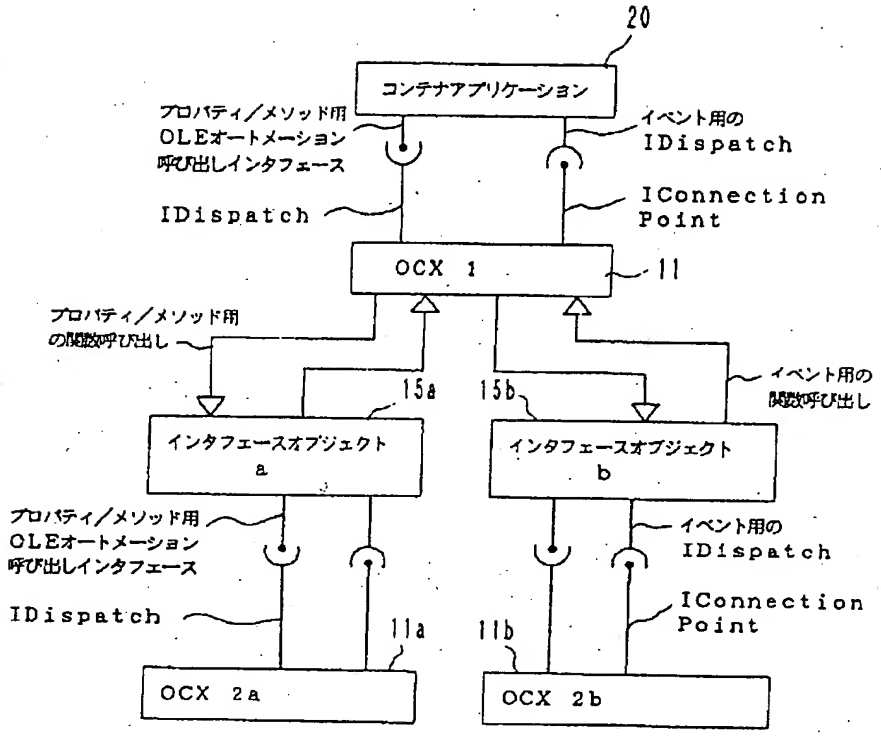
【図３】


start
41
fcntをクリア
42
fcnt ＜ イベント数
N
Y
43
EntryName (fcnt) が
pDispMap () にあるか
N
Y
44
pDispMap () のパラメータ情報が
ParamInfo (fcnt) と
一致するか
N
Y
45
DispIDs (fcnt) をpDispMap ()
でのインデックスに変換するm_dispID配列
を設定する。
46
fcnt <- fcnt +1
イベントに関するコネクションをセットする。
47
48
OK ?
N
Y
49
変換する配列m_dispIDを保存する。
50
正常終了
51
エラー処理

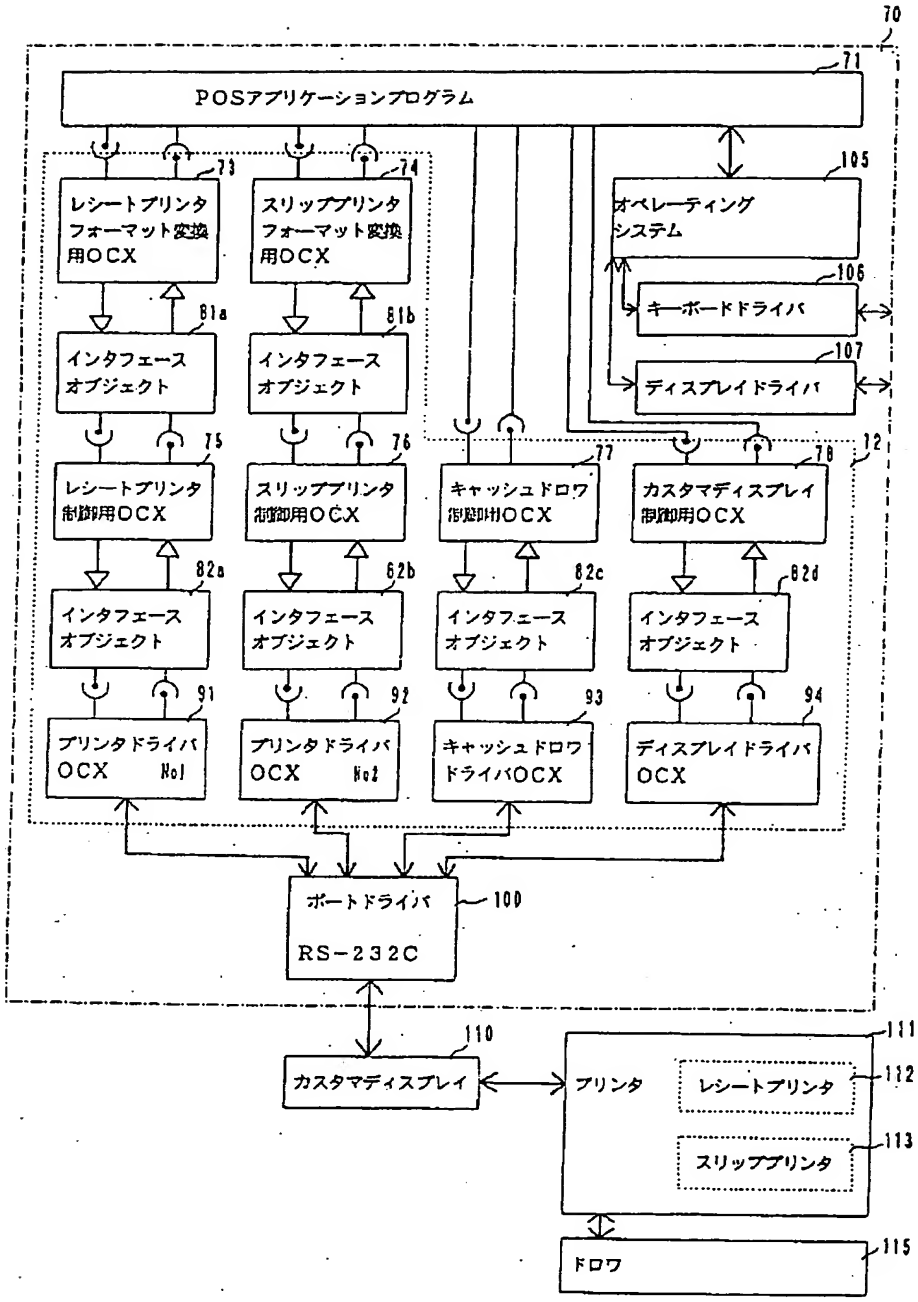
【図４】


start
60
パラメータ正常
N
Y
61
dispIDMemberをpDispMapに合わせて変換する
62
dispIDMemberに対応するイベントハンドラをpDispMapから取り出す
63
イベントハンドラは有効か
N
Y
64
イベントハンドラを呼び出す
正常終了
エラー処理

【図５】


20
コンテナアプリケーション
プロパティ／メソッド用
OLEオートメーション
呼び出しインタフェース
イベント用の
IDispatch
IDispatch
IConnection
Point
OCX 1
11
プロパティ／メソッド用
の関数呼び出し
イベント用の
関数呼び出し
15a
15b
インタフェースオブジェクト
a
インタフェースオブジェクト
b
プロパティ／メソッド用
OLEオートメーション
呼び出しインタフェース
イベント用の
IDispatch
IDispatch
IConnection
Point
11a
11b
OCX 2a
OCX 2b

【図６】


70
71
POSアプリケーションプログラム
73
74
レシートプリンタ
フォーマット変換
用OCX
スリッププリンタ
フォーマット変換
用OCX
105
オペレーティング
システム
106
キーボードドライバ
107
ディスプレイドライバ
81a
81b
インタフェース
オブジェクト
インタフェース
オブジェクト
75
76
77
78
12
レシートプリンタ
制御用OCX
スリッププリンタ
制御用OCX
キャッシュドロワ
制御用OCX
カスタマディスプレイ
制御用OCX
82a
82b
82c
82d
インタフェース
オブジェクト
インタフェース
オブジェクト
インタフェース
オブジェクト
インタフェース
オブジェクト
91
92
93
94
プリンタドライバ
OCX No1
プリンタドライバ
OCX No2
キャッシュドロワ
ドライバOCX
ディスプレイドライバ
OCX
ポートドライバ
100
RS-232C
110
111
カスタマディスプレイ
プリンタ
レシートプリンタ
112
スリッププリンタ
113
ドロワ
115

THIS PAGE BLANK (USPTO)